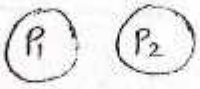
Deadlocks

Definition- If two or more processes are waiting on some event to happens, which never happens, then those processes are said to be involved in the deadlock.

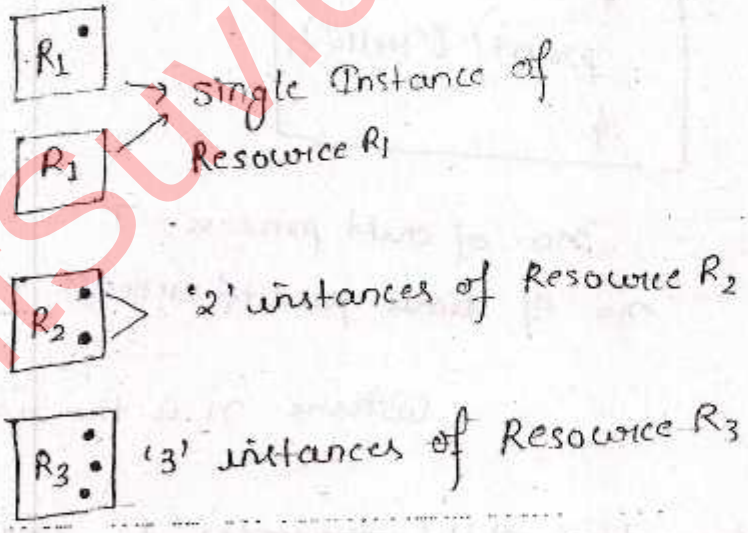
Basics of Deadlock -

Processes



Resources

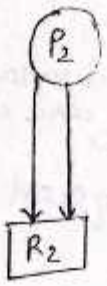
Resources with instances



instances - copy, [here instances of resource R_1, means copy of resource R_1]

Requesting

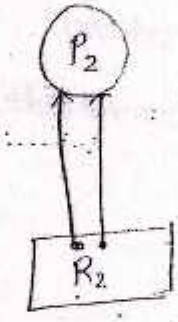
The process P_1 is requesting for one instance of Resource R_1.



The process 'P_2' is requesting '2' instances of resource 'R_2'.

Allocation:

one instance of Resource R_1 is allocated to the process P_1.



'2' instance of Resource R_2 is allocated to process P_2

The resource request and the resource allocation is represented by using the resource allocation graph.

RAG (Resource Allocation Graph) -

It is a graph which consists of set of vertices & set of edges.

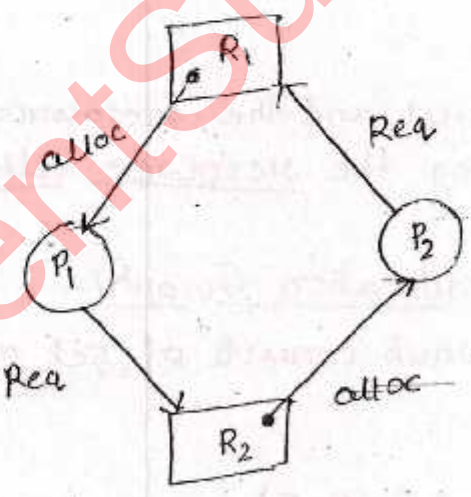
RAG = G(V, E)

- V → Resources, processes
- E → Requesting, Allocation

The Resource Request / Release life cycle (available in the system)

1) The process will request for the resource.
2) Operating System clearly validates the request of the process.
3) If the request made by the process is valid, then the Operating system checks for the availability of the resource.
4) If the resource is freely available then it will be allocated to the process otherwise the process has to wait
5) If all the resources required by the process are allocated then it will go into execution.
6) Once the execution of the process is completed then it will release all the resources.

⟹



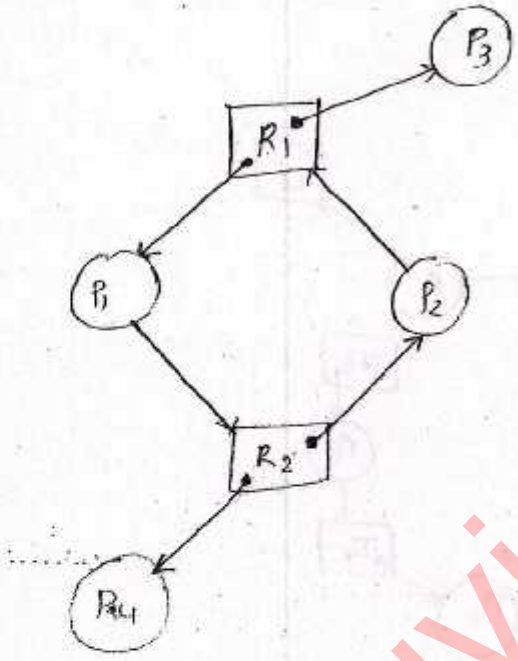
Deadlock

→ Process P_1 is requesting for resource R_2, which is already allocated to process P_2.

→ Process P_2 is requesting for resource R_1, which is already allocated to process P_1

→ Process P_1 can't stop for requesting Resource R_2 & also not releasing the resource R_1 bcz both are required for execution. & similarly process P_2 can't stop for requesting R_1 & also not release the instance of R_2. Both are waiting for an event to occur. Deadlock occurs here.

⟹



No Deadlock

→ P_3 and P_4 are under execution.

→ When P_3 completed its execution, it release the instance of R_1, which is allocated to P_2 & P_2 is going to execute.

→ When P_4 completed its execution. it release the instance of R_2, which is allocated to P_1 & P_1 is going to execute.

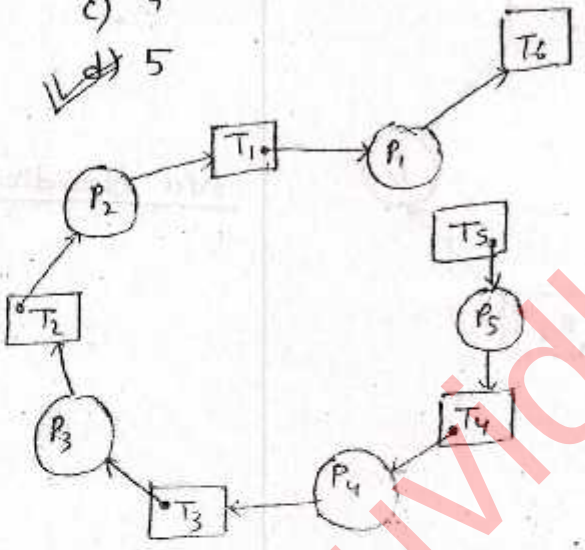
→ All processes are going to execute & complete its execution. No one is waiting forever or wait for an event to occur.

→ Deadlock free.

Q. Consider a system with 'n' processes & 6- Tape drives. Each process requires two Tape drives to complete their execution then what is the max^m value of 'n' which ensures deadlock free operation?

a) 2
b) 3
c) 4
d) 5

Sol:



P_2 - waits - on - T_1 (allocated to P_1)
P_3 - " - on - T_2 (" " P_2)
P_4 - " - " - T_3 (" " P_3)
P_5 - " - " - T_4 (" " P_4)

When P_1 complete its execution by using T_1 & T_6. It will release tape-drive T_1 which is allocated to P_2. After P_2 complete its execution by using T_1 & T_2 & release T_2 - assign to P_3. After P_3 Complete its execution, it will release T_3, which is assigned to P_4. After completion P_4 release T_4 which is allocated to P_5. P_5 complete its execution by using T_5 & T_4 It become deadlock free.

Q. Consider a system with the '3' processes P_1, P_2 & P_3 each process requires two units of resource R to complete their execution. Then what is the min^m no. of resources require to ensure deadlock free oper^n?

Sol: P_1 P_2 P_3 (if 3 resources, then all P_1 P_2 P_3 are waiting for 2nd unit of resource)
1 1 1
Deadlock

P_1 P_2 P_3 (Resource = 4) Deadlock free
1 1 1
1

Q. consider a system with 'n' process & 6- Tape drives. If each process require 3- tape drives to complete its their execution, then what is the max^m value of 'n' which ensures deadlock free oper^n?

Sol:
P_1 P_2
1 1
1 1
1 1
Deadlock free

P_1 P_2 P_3
1 1 1
1 1 1
Deadlock (all are waiting for third resources)

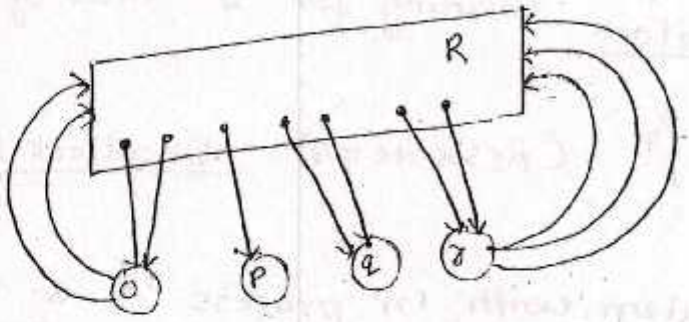
Q. Consider a system with three processes P_1, P_2 & P_3. The peak demand (max^m) of each process for a particular resource type 'R' is 5, 9, 13 respectively. Then what is the min^m no. of resources required to ensure deadlock free oper^n?

Sol:
$P_1 - 5 - 4$ — if allocate 1 more resource, it became deadlock free.
$P_2 - 9 - 8$
$P_3 - 13 - 12$
$24 + 1 = 25$

1) Necessary condition → Deadlock may be possible or may NOT be possible.

2) Sufficient condition → Deadlock surely or never be possible.
↳ least upper bound which satisfies condition is called as sufficient condition.

49 Workbook Q.15



n = 4 (P, P, Q, r)

$x_P = 2$

$x_P = 1$

$x_Q = 2$

$x_r = 2$

1. Deadlock characteristics (Some cond^n, if they are satisfied mean deadlock occur)
2. Deadlock prevention (to prevent deadlock to be occur)
3. Deadlock Avoidance (after deadlock occurs, try to near about to be occur of deadlock, try to avoid the cond^n that lead to deadlock) — V. Imp *
4. Deadlock Detection
5. Deadlock Recovers

Deadlock Characteristics

1) Mutual Exclusion
2) Hold and wait
3) No pre-emption
4) Circular wait

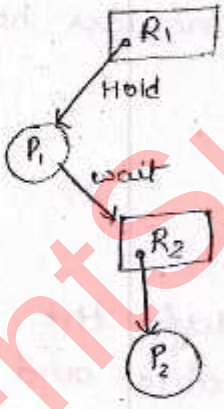
1) Mutual Exclusion :

⇒ The resource has to be allocated to only one process or it is freely available.

⇒ There should be a one-to-one relationship b/w the resource and the process.

2) Hold and wait:

⇒ The process is holding the resource and waiting on some other resource simultaneously.



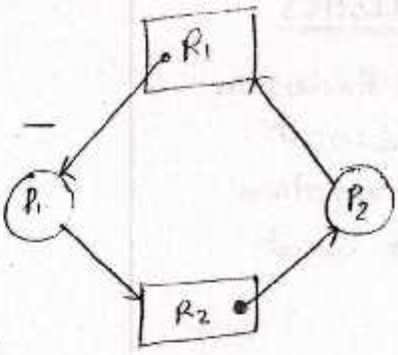
3) No-preemption :

⇒ The resource has to be voluntarily released by the process after completion of execution.

⇒ It is not allowed to forcefully pre-empt the resource from the process.

4) Circular wait:

The processes are circularly waiting on each other for the resources



$R_1 \rightarrow P_1 \rightarrow R_2 \rightarrow P_2$

Circular wait

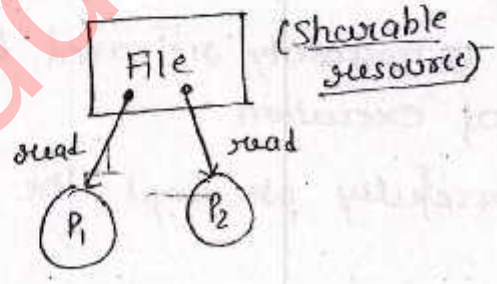
Note: If all the above four conditions are existing simultaneously in the system then definitely there exist a deadlock. (not only 1, 2 or 3, all the above - Deadlock)

⇒ All the above four conditions are not truely independent because the circular wait includes hold and wait.

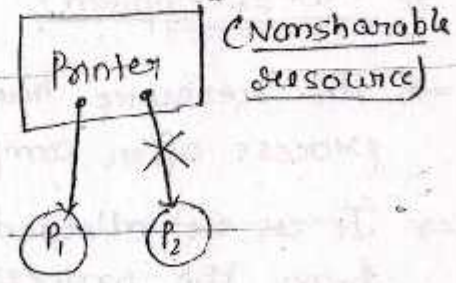
Deadlock Prevention

1) Mutual Exclusion:

It is not possible to dissatisfy the mutual exclusion always because of sharable and non-sharable resources.



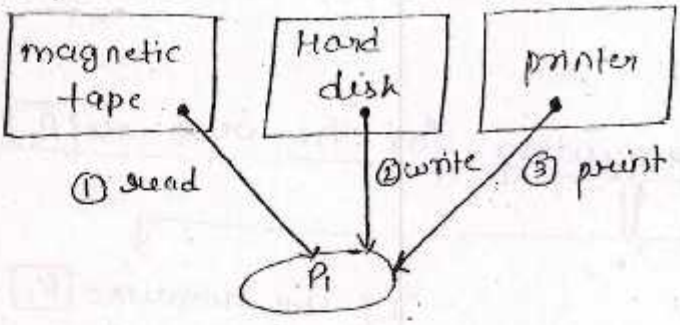
Dissatisfy Mutual Exclusion Condition



Satisfy Mutual Exclusion Condition

2) Hold and wait:

(i) Allocate all the resources required by the process before start of execution.

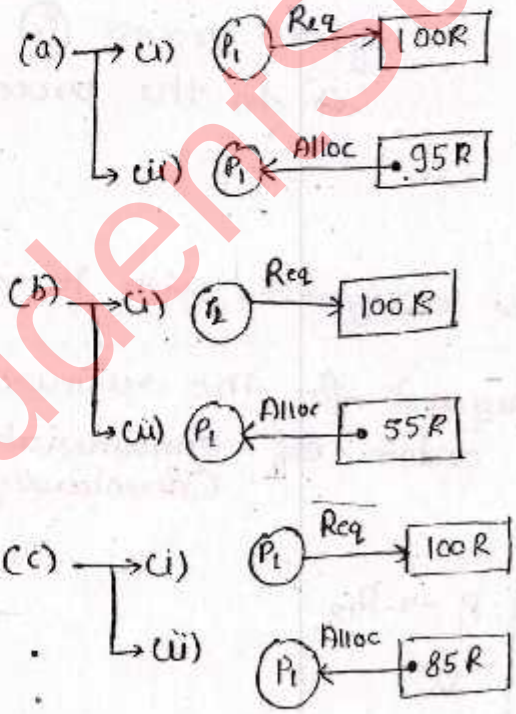


problem - low device utilization

Unnecessary P_1 is holding the printer, eventhough it is not require to P_1 at the stard of execution. bcz P_1 require printer at the end of execution.

(ii) The process should release all the existing resources before making the new request.

The process P_1 require 100R to complete the execution:



Starvation

⇒ If the above condition is hold to dis-satisfy hold and wait then it is possible for the processes to go into starvation.

3) No pre-emption :

The process (P_1) is requesting for the resource $[R_1]$

If the resource $[R_1]$ is free, then it will be allocated to process (P_1)

If the resource $[R_1]$ is NOT free, and it is allocated to the other process (P_2)

If the process (P_2) is in execution then the process (P_1) has to wait

The process (P_2) is NOT in the execution, and it is waiting some other resource $[R_2]$

pre-empt the resource $[R_1]$ from process (P_2) & allocate it to the process (P_1)

4) Circular Wait:

⇒ The resources will be assigned with the numerical no.

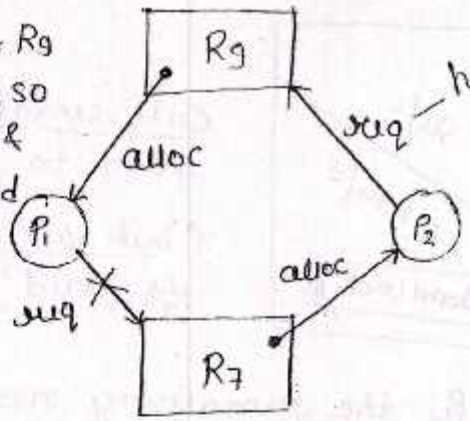
⇒ The process can request for the resource only in the increasing order of enumeration. (numbering)

$P_1 \rightarrow R_9$ ✓ (allowed)

$P_1 \rightarrow R_7$ ✗ (not allowed)

$P_1 \rightarrow R_{12}$ ✓ (allowed)

(P_1 request for R_9 it is first, so allowed & R_9 is allocated to P_1)



(P_2 request for R_7, it is allowed & R_7 is allocated to P_2)

Deadlock Avoidance

Deadlock avoidance can be implemented by using Bankers algorithm.

Bankers algorithm having parameters:

Total available

A	B	C
10	5	7

	max need			current allocation			current available (3 3 2)			Remaining need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0 →	7	5	3	0	1	0	3	3	2	7	4	3 – P_0
P_1 →	3	2	2	2	0	0	5	3	2	1	2	2 – P_1
P_2 →	9	0	2	3	0	2	7	4	3	6	0	0 – P_2
P_3 →	2	2	2	2	1	1	7	5	3	0	1	1 – P_3
P_4 →	4	3	3	0	0	2	10	5	5	4	3	1 – P_4
				7	2	5	10	5	7	always equal to total available		

(Total current allocation)

Remaining need = Max need − current allocation

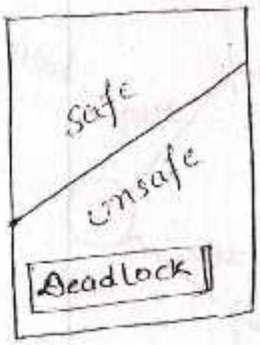
Safe sequence

P_1, P_3, P_0, P_2, P_4

after P_1 completed its execution it released the currently hold resources.

now,

current available = (332 − 122) + 122 + 206 = 532



(all unsafe are not said to be deadlock)

(but all deadlock is said to be unsafe)

1) If we can satisfy the remaining need of all the processes with the current available resources, then the system is said to be in safe state. Otherwise the system is said to be in the unsafe state.

2) If the system is in the unsafe system, then it is possible for the deadlock.

3) The order in which we satisfy the remaining need of all the process is called as safe sequence.

4) The safe sequence may not be unique, we can have the multiple safe sequence.

5) The unsafe state purely depends on the behaviour of the processes. (unpredictble)

6) Each and every time, when the process is requesting for any resource, the Bankers algo will be used to identify whether the system is in the safe state or unsafe state.

7) If the system is in the safe state then the request of the process will be granted, and if the system is in the unsafe state, the request of the process will be denied.

Workbook 6.3 P-47.

	Max need	Current allocated	Current available	Remaining need
P_0	6 5 4	0 3 4	4 3 1	6 2 0 → P_0
P_1	3 4 2	2 1 2	6 4 3	1 3 0 → P_1
P_2	1 0 4	0 0 2	6 7 7	1 0 2 → P_2
P_3	3 2 5	1 2 1	6 7 9	2 0 4 → P_3
			7 9 10	

I. Safe sequence - P_1, P_0, P_2, P_3

II. Safe sequence - P_1, P_2, P_0, P_3

III. Safe sequence - P_1, P_3 X

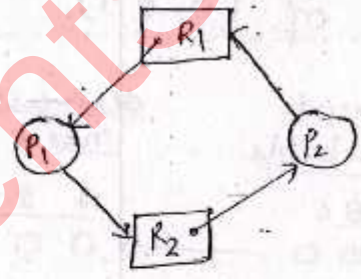
IV. Safe sequence - P_1, P_2, P_3, P_0

IV	II	III
4 3 1	4 3 1	4 3 1
6 4 3	6 4 3	6 4 3
6 4 5	6 4 5	
7 6 6	6 7 9	
7 9 10	7 9 10	

Ans. (a)

Deadlock Detection

(i) The Resources are of single Instance type:



Run: Cycle Detection Algorithm

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_2 \rightarrow P_1$

Cycle exist

means deadlock exist when but only, resources are of single instance type.

⇒ If all the resources are of single instance type then the cycle in resource allocation graph is necessary and sufficient condn for occurrence of deadlock.

⇒ If all the resources are of not single instance type, then the cycle in the resource allocation graph is just a neccesary condition but not sufficient condition for occuring of deadlock.

(ii) The Resources are of multiple instance type:-

	current (Resource) allocation			Current available			Remaining need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	0	1	0	2	0	2
P_2	3	0	3	3	1	3	0	0	0
P_3	2	1	1	5	1	3	1	0	0
P_4	0	0	2	7	2	4	0	0	2
				7	2	6			

Safe Sequence- P_0, P_2, P_1, P_3, P_4

⇒ What happens? if the process P_2 is requesting for additional resources of (0 0 1)

	current allocation			Current available			remaining need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0 ← P_0
P_1	2	0	0	0	1	0	2	0	2 ← P_1
P_2	3	0	3				0	0	1 ← P_2
P_3	2	1	1				1	0	0 ← P_3
P_4	0	0	2				0	0	2 ← P_4

(can't move further as not able to satisfy any need of further processes)

Compleate its execution → P_0 (not in deadlock) ⇒ unsafe ⇒ Deadlock (P_1, P_2, P_3, P_4)

⇒ Unsafe state is unpredictable & it is purely depend on behaviour of processes means changing nature of their remaining need. If remaining need is constant (not changing) surely deadlock occurs. But if the remaining need is changes it may come again in safe state.

Deadlock Recovery:

→ No traditional OS used this deadlock algorithm bcz it is time consuming & takes more space.

→ Real time OS uses this deadlock algorithm.

i) Killing the process –

a) Kill all the processes which are involved in the deadlock. (User processes)

b) Kill one after the other (one-by-one)
(Kill one process which is involved in the deadlock, & again check for deadlock by using deadlock detection, if processes are deadlock exist then again apply ~~dead~~ kill one more process & again apply deadlock detection & so on.

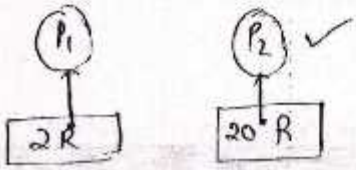
On what basis we first delete a process – (Criteria) (Kill)

→ low priority

→ % of process completion

Kill P_1 first.

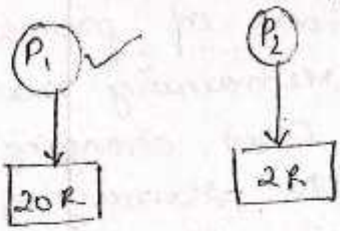
→ # of resources, the process is holding



Better to kill P_2, it release 20 Resources.

→ # of resources the process is Requesting

(Better to kill P_1 first)



2) Resource Preemption -

→ The resource will be preempted from the processes which are involved in the deadlock, and the pre-empted resources will be allocated to other processes, so that there is a possibility of recovering the system from the deadlock.

→ If the resource pre-emption is followed to recover system from the deadlock, then it may be possible for the processes to go into starvation.

3) Ostrich algorithm

→ ignore the deadlock